OWASP ZAP in CI/CD

- **Why it Matters**:
    - Integrating ZAP in CI/CD pipelines ensures security is continuously validated, alongside functional tests, for every code change.
    - Automating these scans reduces manual intervention, ensuring **early detection** of potential vulnerabilities, preventing them from reaching production.

▼ Step 1 - Setting Up OWASP ZAP on GitHub Actions

1. Go to your GitHub repository and select the **Actions** tab.
2. Choose to create a **new workflow** or **start from scratch**.
3. If available, use **OWASP ZAP Baseline Scan** from the workflow options, or add the following YAML file:

YAML Configuration Example:

```
name: OWASP ZAP Scan

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Build with Maven
        run: mvn clean install

      - name: Build Docker Image
        run: docker build -t hackapp .

      - name: Run Docker Container
        run: docker run -d -p 8080:8080 hackapp

      - name: OWASP ZAP Scan
        uses: zaproxy/action-full-scan@v0.11.0
        with:
          target: 'http://localhost:8080'
          docker_name: 'ghcr.io/zaproxy/zaproxy:stable'
```

Additional Tips:

- Ensure **Docker** is correctly set up in your project for starting the `HackApp` or any other service to scan.
- Test locally before committing to ensure smooth integration with GitHub Actions.

▼ Step 2 - Configure the Workflow

Key Configurations:

1. `Target URL`: Replace `http://localhost:8080` with your app's correct URL (make sure the app is running at this URL when the scan starts).
2. `failOnWarning`: Set to `false` initially to gather feedback without blocking the CI pipeline. Once stabilised, switch to `true` to block merges if ZAP finds critical vulnerabilities.
3. `GITHUB_TOKEN`: This environment variable is used to authenticate the GitHub Action for creating reports or raising issues. Ensure it's securely stored in the repository's secrets.

Example Configurations:

- `timeToWait`: If your app takes time to start (especially with Docker), increase `timeToWait` to allow ZAP more time before beginning its scan.

```
with:
  target: http://localhost:8080
  timeToWait: 60
```

▼ Step 3 - Running ZAP in the CI Pipeline

Execution:

- On every **push** or **pull request**, OWASP ZAP will automatically scan the application.
- **Types of Scanning**:
  - **Passive Scanning**: Default scan that does not alter the application; checks security headers, cookies, etc.
  - **Active Scanning**: (Optional, but recommended in later stages) Actively attempts attacks on the app (e.g., SQL injection, XSS) to test for exploitable vulnerabilities.

Important Tips:

- Run the **baseline scan** first to identify passive issues (headers, SSL/TLS problems). Add **active scans** when your pipeline is more mature and stable.
- **Alerts**: Ensure your development team understands ZAP's report format and prioritizes **critical** and **high** alerts for fixing immediately.

▼ Step 4 - Integrating Dynamic Testing with OWASP ZAP

Why Integrate Dynamic Testing?:

- Early detection of security vulnerabilities in staging ensures the issues don't leak into production.
- Continuous security ensures each code change is tested in CI/CD, giving real-time feedback to developers.

Steps to Integrate:

1. Add the **OWASP ZAP GitHub Action** to your pipeline.
2. Ensure ZAP runs as part of your **pull request checks**.
3. Tune **thresholds** to manage false positives:
    - Start with **informational alerts** off.
    - Raise alerts to block merges only for **high** or **critical** issues.

Example:

```
with:
  target: http://localhost:8080
  failOnWarning: true
  cmdOptions: '-config scanner.alertOn=High'
```

This example sets ZAP to fail the pipeline if **high-risk** vulnerabilities are found.

▼ Running OWASP ZAP with GitHub Actions - Reports & Metrics

Key Metrics:

- **Vulnerabilities Found**: ZAP classifies risks into different levels like **informational**, **low**, **medium**, and **high**.
- **Risk Levels**:
    - **High**: Issues that could lead to severe breaches, like SQL Injection.
    - **Medium**: Vulnerabilities that could be exploited, but require user interaction or specific conditions.
    - **Low**: Best practice violations or less likely exploits.

Example Report Output:

- **Alerts**: SQL Injection, XSS.
- **Best Practice Violations**: Missing security headers like **CSP**, **X-Frame-Options**.

Tip: Regularly review the reports and fix **high** and **critical** issues promptly. Add lower-level issues to the backlog to be addressed over time.

▼ Customising OWASP ZAP for Your CI Pipeline

Tailoring OWASP ZAP to Your Needs:

- **Custom Alert Thresholds**:
  - Focus on **critical** or **high** vulnerabilities to reduce noise.
  - Set thresholds to block merges only if severe issues are detected:

    ```
    with:
      target: http://localhost:8080
      failOnWarning: true
      cmdOptions: '-config scanner.alertOn=High'
    ```

- **Custom Reports**: Use the ZAP add-ons to customize reports and generate more detailed insights on vulnerabilities:
  - Example: Use **Alert filters** to suppress or prioritize certain alerts.
- **ZAP's API**: Leverage the API to automate more complex workflows, such as scanning specific endpoints, testing with authentication, or setting up advanced alerting.

▼ Best Practices for Integrating OWASP ZAP

Suggestions:

1. **Prioritise Critical Alerts**: Initially, focus on **high-risk** vulnerabilities. Set rules to fail the build only for these to avoid alert fatigue.
2. **Handling False Positives**: Regularly review and adjust ZAP's configurations to suppress non-critical issues:
   - Use **alert filters** to ignore low-priority findings (e.g., missing `X-Frame-Options` for non-UI APIs).

3. **Continuous Improvement**: Regularly review ZAP configurations and adapt them to the evolving architecture of your application. Ensure new vulnerabilities are detected and addressed as part of your development cycle.